Artificial Intelligence Project--RLE and MIT Computation Center
Memo 47--

32p.

A Proposal to Investigate the Application of a Heuristic Theory of Tree-searching to a Chess Playing Program

by

Burton H. Bloom

February 15, 1963

"Problems are posed by fools like me,
But only Heuristics can search a tree."
--with apology to Joyce Kilmer.

Introduction

The problem of devising a mechanical procedure for playing chess is fundamentally the problem of searching the very large move-tree associated with a chess position. This tree-searching problem is representative of a large class of problems. Consequently, we will first present briefly a general theory of tree-searching problems. This theory will be useful in clarifying the intention of our proposed research.

The proposed project will consist of four phases.

Phase 1: Designing and programming the heuristic procedure.

Phase 2: Using the program in an experimental environment.

Phase 3: Evaluating the experimental results.

Phase 4: Preparing the final report on the research.

After the preliminary presentation of theoretical background, we will present a discussion of each of the four phases.

It is intended that the final report will be the author's doctoral dissertation.

Descriptive Theory

In this section, we will present a theory of tree-searching problems divorced from the methodology of search. That is, we will discuss the "tree" and "problems" part of "tree-searching problems". These two parts will be discussed first separately, and then with respect to their relationships to each other.

Trees

A tree is a special kind of finite lower semi-lattice. The specific kind it is will be described later, after a suitable nomenclature has been presented. The concept of a lower semi-lattice

is only used for its descriptive application to the nomenclature being presented.

The elements of the tree (i.e., of the lower semi-lattice) are called <u>nodes</u>. The <u>base</u> of the tree is that node less than all other nodes. An <u>extremity</u> or <u>terminal</u> node is any node such that there is no node greater than it. A <u>branch-segment</u> is any string of nodes, $a_1, a_2, \ldots, a_n$ having the relationship $a_1 < a_2 < a_3 < \ldots < a_n$ and satisfying the condition that $a_1 < b < a_n$ implies $b = a_i$ for some $i = 2, 3, \ldots, n-1$. A <u>branch</u> is a branch-segment whose greatest node is an extremity. A <u>path</u> is a branch whose least node is the base. An <u>α-branch</u> is any branch whose least node is the node α. Thus if a_1 represents the base, then any a_1-branch is a path. The <u>α-path</u> is the branch segment whose least node is the base and whose greatest node is the node α. Thus if, a_k is an extremity, then the a_k-path is a path. The <u>length</u> of a branch segment is the number of nodes in the branch segment. For any node α not the base, the <u>ply</u> of the node α is the length of the α-path. For completeness, the ply of the base is one.

In these terms, a <u>tree</u> is a lower semi-lattice such that every node belongs to at least one path. Thus, for any node α not the base, there is a unique α-path. For any extremity α there is a unique path terminated by α.

Consider a non-terminal node α. Let the ply of α be n. Let $A_\alpha = \{a_1, a_2, \ldots, a_n\}$ be the set of nodes greater than α with ply $n+1$. The set A_α is called the <u>set of alternatives-at-α</u>, and each a_i is called an <u>alternative-at-α</u>.

For clarity in later discussions we will designate a tree by

T. The set of nodes of T will be designated by A. The individual nodes of T will be designated by a_i. That is $A = \{a_1, a_2, \ldots, a_n\}$. The base of T is designated by a_1. We will also assume that for two nodes of T, a_i and a_j, $i > j$ implies ply $(a_i) \geq$ ply (a_j).

We also define a <u>sub-tree</u> of T, denoted T , as that portion of T whose nodes are all $\geq \alpha$. Thus is the base of the sub-tree T_α. We also observe that an α-branch of T is a path of T_α. Also, $T = T_{a_1}$.

By way of clarification, let us consider the game of chess in terms of the above nomenclature. Let us assume we are confronted with a chess position, and we wish to decide the move we will make. This given position corresponds to the base a_1 of T. The other nodes of the tree correspond to the positions which result from various legal sequences of moves. In the given position there are a finite number of legal moves we can make. Each legal move will result in a new position. Each of these new positions corresponds to an alternative at a_1. The set of new positions corresponds to A_{a_1}. In the course of deciding on our move, we will analyze a number of variations. A variation is a legal sequence of moves. The moves in sequence result in a sequence of positions corresponding to a sequence of nodes of successive plys. If a_k corresponds to the end position of the variation, then the variation corresponds to the a_k-path. Let us consider a position which can be legally reached from the given position and which is a clearly won, lost, or drawn position. That is, the position is checkmate,

stalemate or some other legally drawn position. Then this position corresponds to an extremity of T. The sequence of positions leading to this position by a sequence of legal moves corresponds to a path of T.

In considering the game of chess in these terms, there are a few minor difficulties. For example, it is possible in chess that the same position can be reached by more than one sequence of moves. There is also the problem concerning the rule of draw by repetition of moves. These difficulties are in fact easy to overcome, and in the interest of brevity we will not elaborate here on their nature and solution.

Problems

Let us now consider what we will mean by a well-defined class of problems for tree-searching. We assume the existence of a problem space S and a finite transformation space F. Each element f of F has a subset D_f of S as a domain and a subset R_f of S as a range. We consider the mapping φ which maps each element $s \in S$ onto its set of applicable transformations, i.e. those transformations which have s as an element of their domains. That is $\varphi(s) = \{f \mid s \in D_f\}$. We denote this set by L_s. In these terms, a well-defined class of tree searching problems is defined by two given well-defined spaces S and F and an effectively computable given mapping φ. It is also necessary that the elements of F be effectively computable transformations on the elements $s \in S$ in their domains. Let $\bar{S}$ denote that subset of S such that $s \in \bar{S}$ implies $\varphi(s)$ is the empty set, i.e. $\bar{S}$ is the nullity of φ.

A well-defined specific tree-searching problem belongs to a well-defined class, and is "properly defined" in terms of the three following constituents:

1) A positive integer N;

2) An element s_1 in S;

3) A finite subset $S' \subset S$.

"Properly defined" means that the three constituents are related in the following way.

A well-defined problem consists of finding a "suitable" finite sequence of transformations:

$$f_1, f_2, \ldots, f_k;\ f_i \in F, (i = 1, 2, \ldots, k);\ k \leq N.$$

The sequence is "suitable" if it satisfies the conditions:

$$s_1 \in D_{f_1}$$

$$f_1(s_1) \in D_{f_2}$$

$$f_2 \cdot f_1(s_1) \in D_{f_3}$$

$$f_{k-1} \cdots f_2 \cdot f_1(s_1) \in D_{f_k}$$

$$f_k \cdots f_2 \cdot f_1(s_1) \in S'$$

That is the application of the transformations $f_1, f_2, \ldots, f_k$, in sequence to the element s_1, results in an element of S'.

We will now discuss the game of chess in terms of a well-defined tree searching problem as presented above. We again assume that we are confronted with a position and are trying to decide on our move. The presented position corresponds to s_1. The set of all possible chess positions corresponds to S. The set of all possible moves constitute the set F. (Here we might assume that there are 4096 possible moves, where a move consists of mov-

ing from a square to a square. The unusual moves of castling and capturing en-passant can also be suitably defined in these terms.) φ corresponds to a mapping from chess positions to the legal moves which can be made in those positions. N corresponds to some maximum number of half-moves required to play the longest game. The rules of chess guarantee the existence of a finite N. In playing chess one in general tries to obtain a win or a draw. From some positions only a win is acceptable. Consequently the set S^1 usually corresponds to the set of wins and draws, and sometimes only to the set of wins. In this manner we consider the set $S^{\#}$ to be the set of all favorable mates and perhaps draws. Thus we consider the problem of playing chess as that of finding a sequence of moves leading to a win or perhaps draw. Of course, this does not take into consideration that our opponent isn't helping us in our enterprise. It is for this reason the formulation begins to be somewhat inadequate. However, it will be shown later that when methodology of tree-searching is considered, this difficulty (of considering a competitive situation in terms of a well-defined tree-searching problem) becomes unimportant. Indeed, when we discuss the heuristic theory of tree-searching problems this difficulty will completely vanish. For the purpose of this section of the discussion, however, it suffices if we consider that the mapping φ is so constituted that the sets L_{s_i}, corresponding to the sets of our opponents possible moves at his turns to play, contain only the moves he would in fact make, or the sets of moves we could reasonably expect him to make. This, of course, causes φ to be no longer effectively computable in the same sense as it is for gener-

ating the legal moves for each position.

In summary, we observe that the class of problems expressable in terms of our descriptive theory is not general enough to include chess without radical re-interpretations. We anticipate what is to follow by noting that this limitation is completely unimportant when we consider methodology of search in general, and the limitation vanishes when we consider heuristic search methodology in particular.

Trees and problems

In this subsection we will present the interrelationships between the theory of trees and the theory of well-defined problems for tree-searching. Let us consider a well-defined problem characterized by S, F, φ, $\bar{S}$, S', s_1, and N. S is the problem space. F is the set of possible transformations which transform one element of S into another. φ is a mapping from S into the set of subsets of F. S' and $\bar{S}$ are subsets of S such that $S' \subset \bar{S} \subset S$. Also, s_1 is an element of S, and N is and integer. We consider a tree T constructed as follows. s_1 corresponds to the base a_1. Consider $\varphi(s_1) = L_{s_1} = \{f_2, f_3, \ldots, f_k\}$. Now consider the set s_1 of elements of S: $s_i = f_i(s_1)$, $(i = 2, 3, \ldots, k)$. These elements of S correspond to the nodes a_i of T which are alternatives-at-a_1 (i.e. the set A_{a_1}) and which comprise the nodes of ply two. Proceeding in a similar manner from each of the s_i, $(i = 2, 3, \ldots, k)$, we obtain elements of S which respectively correspond to the sets of nodes A_{a_i}. $\bigcup_{i=2}^{k} A_{a_i}$ is the set of all nodes of ply three. By

applying this process repeatedly, at most N times, we will generate all of the nodes of T. No element of T will have a ply greater than N+1.

When an element of $\bar{S}$ is obtained, in the process of "growing" the tree, its corresponding node will be an extremity. In addition to these extremities there will be all nodes of ply N+1. Consequently, the set of extremities of the tree will consist of those nodes corresponding to elements of $\bar{S}$, and those nodes of ply N+1. Thus, the problem of finding a suitable sequence of transformations is the same as that of searching the tree for a "suitable" path. A path is "suitable" if its extremity corresponds to an element of S'.

We now amend our nomenclature as follows. We denote by A' the subset of A whose elements correspond to elements of S'. The set of terminal nodes of T we denote by $\bar{A}$. In these terms, the tree-searching problem is that of finding a path of T terminated by an element of A'.

Tree-searching methodology

Recursive statement of problem

Let us consider a tree-searching problem characterized by sets S and F, a mapping φ, subsets $S' \subset S \subset S$, an element $s_1 \in S$, and an integer N. Let the associated tree be T having a set of nodes $A = \{a_i\}$, $(i = 1, 2, \ldots, n)$, where a_1 is the base of the tree and corresponds to s_1. For each element of A, a_i, let s_i, denote the corresponding element of S. The tree-searching problem may be stated as the problem of finding a path in T terminated by an

element of A^*. We will call such a path a solution-path of T.

We may also state the problem recursively as follows. Given any subtree T_α such that there exists a solution-path, to find an alternative-at-α, say β, such that the sub-tree T_β also has a solution path. It is obvious that if we can solve the recursively stated problem, we can also solve the originally stated problems.

Algorithmic tree-searching

There are several algorithms for searching a tree. The most straightforward one consists of always choosing the alternative-at-α with the next highest index. Continue selecting alternatives-at-α in this fashion until either a solution path is found, or until all alternatives are exhausted. If all alternatives are exhausted, then T has no solution-path. In this case, choose the next highest indexed alternative-at-β where α is an alternative-at-β. This algorithmic procedure will obviously try all possible paths in the tree systematically until one is found terminated by an element of A^*.

Let us restate this algorithm in a slightly different form. For this purpose we will define two classes of nodes, C_1 and C_2, such that:

1) $A^* \subset C_1$

2) for any node α, if β is an alternative-at-α, and $\beta \in C_1$, then $\alpha \in C_1$

3) $C_2 = A - C_1$

The algorithm is now stated as follows. For any subtree T_α, try in some order the various alternatives-at-α until an element of C_1 is found among these alternatives. Then $\alpha \in C_1$. If the set of

alternatives-at-α is exhausted without finding one belonging to C_1, then $\alpha \in C_2$.

We note that for any interesting problem, the above method is completely inadequate, since the tree will be too large to search in this fashion in any reasonable amount of time.

We will now reconsider chess in terms of this algorithm to show that the previously indicated inadequacy of the descriptive theory (i.e, the difficulty of considering competative situations as trees) is no longer important when methodology is considered.

We again consider we are given a position s_1 corresponding to the base a_1 of the tree T. The nodes of T are elements of $A = \{a_i\}$, $(i = 1, 2, \ldots, n)$. Let the element of S corresponding to each node a_i of T be denoted by s_i. Let S* denote the set $\{s_i\}$, $(i = 1, 2, \ldots, n)$. We now consider three subsets of S, namely S_1, S_2, and S_3, which are the sets of wins, draws, and losses respectively. Let A_1, A_2, and A_3 correspond respectively to $S_1 \cap S^*$, $S_2 \cap S^*$, and $S_3 \cap S^*$. We now define three classes of nodes, C_1, C_2, and C_3, recursively as follows.

1) $A_1 \subset C_1$.

2) $A_3 \subset C_3$.

3) For any node α of odd ply, $\alpha \in C_1$ if there exists an alternative-at-α, say β, such that $\beta \in C_1$.

4) For any node α of even ply, $\alpha \in C_3$, if there exists an alternative-at-α, say β, such that $\beta \in C_3$.

5) For any node α of even ply, $\alpha \in C_3$, if all alternatives-at-α are in C_3.

6) For any node α of odd ply, $\alpha \in C_1$, if all alternatives-at-α are in C_1.

7) $C_2 = A - C_1 - C_3$. Note, $A_2 \subset C_2$.

Now our algorithm is stated as follows.

For any sub-tree T_α, try in some order the alternatives-at-α.

If α is of odd ply, the procedure is:

1) Accept the first alternative-at-α of class C_1. Then α is of class C_1.
2) If there are no alternatives-at-α of class C_1, accept any of C_2. Then α is of class C_2.
3) Otherwise α is of class C_3.

If α is of even ply, we have the parallel procedure with C_1 and C_3 interchanged.

We thus see that when we consider the above recursive procedure for searching a tree, chess is not significantly different from that for searching the non-competitive trees. The procedure is always to choose an alternative which is "acceptable", where "acceptability" is recursively determined by the procedure. The only difference is in the specification of the classes, in terms of which the recursive process is defined.

We do note however that the amount of a tree that must be searched in order to find a path is much smaller for non-competitive trees, as compared to chess-type trees. This is because for the former the solution is determined as soon as any element of A' is found. That is, in general it is not necessary to try all alternatives before selecting an alternative. On the other hand, in chess-type trees, it is in general necessary to consider all alternatives.

This point can be clarified by considering a "lucky" tree so constructed that the first alternative to be tried at each node is acceptable, if any are acceptable. For non-competitive trees, this would mean that the first path tried would be a solution. However, for chess-type trees, it would still be necessary to search considerably more of the tree in order to determine that all the nodes of the first path were in fact "acceptable" alternatives. It is easy to see that for such "lucky" competitive trees having the same number M of alternatives at each node, and each path of length 2N+1, the portion of this tree of M^{2N} paths that must be searched is $2M^{N} - 1$ paths. (See [1].) However, we do not normally expect to be "lucky" enough to avoid searching a good portion of a tree in any algorithmic procedure applied to any kind of interesting tree-searching problem.

Heuristic Tree Searching

We here present a heuristic theory of tree searching expressed in terms of two classes of heuristics. These classes will be called the class of meta-heuristics and the class of special-heuristics. The meta-heuristics are heuristics applicable to super classes of well defined tree-searching problems, while the special-heuristics are those applicable only to a particular class of tree-searching problems. An example of a meta-heuristic is the alpha-beta heuristic [1], which is applicable to searching any competitive game type tree. An example of a special-heuristic is: "Always check, it might be mate." This special heuristic is of course, only applicable to the class of tree-searching problems associated with chess.

The theory to be explored in the proposed search involves

a particular set of six meta-heuristics and the manner in which these meta-heuristics are related to the special heuristics. These six meta-heuristics we will designate respectively as: Evaluation, Selection, Summary, Look-ahead, Redundancy, and Planning. We now present a brief discussion of each of these meta-heuristics.

1. Evaluation

This meta-heuristic is that of establishing (and using) an evaluation function which maps the space of problems S into a space of values V. The space V may be scalar, vector, or non-numeric. No matter what the form of V, its elements are to be considered as a "measure of goodness" for the corresponding elements of S. The particular form of V, and the evaluation mapping from S into V, will be described by a set of special-heuristics suitable to the particular problem space S. We will call this meta-heuristic Evaluation, and we will call the particular special-heuristics, which define the mapping from S into V, as the evaluators, or evaluation-functions.

2. Selection

This meta-heuristic is that of extablishing (and using) a selection function (plausible-move-generator) which determines for each α which of the alternatives-at-α are to be explored, and in which order they are to be chosen for exploration. Evaluation may be used to assist in effecting this heuristic. This meta-heuristic we will call Selection, and the particular special-heuristics used in Selection we will call the selection-functions or plausible-move-generators. We note that the ply of α , as well as the information

gathered by exploring some of the alternatives-at-α , may be used as arguments to the various selection-functions.

3. Summary

This meta-heuristic is that of establishing (and using) a function which maps the set of subsets of V, the space of evaluations discussed under Evaluation, into V. The purpose of this meta-heuristic is to provide a means of recursively assigning a value (element of V) to a node α as a summary of the values assigned to the alternatives-at-α . We will call this meta-heuristic Summary. The form of Summary almost exclusively used in searching a competitive game tree is the meta-heuristic commonly known as min-max, i.e. maxmizing at odd plys, and minimizing at even plys. We note that the value assigned to a node α , for which Selection has not provided any alternatives-at-α to be explored, will be the value determined by Evaluation.

4. Look-ahead

This meta-heuristic is that of organizing Evaluation, Selection and Summary in a particular way in order to achieve a general heuristic procedure for tree-searching. We recall that the problem of searching a tree involved choosing an "acceptable" alternative-at-α for each α , where "acceptability" was determined recursively. This recursive determination of "acceptability" involved exploring a large portion of T_α to terminal nodes. What is proposed here is the use of Evaluation, Selection and Summary to recursively select an "acceptable" alternative-at-α by exploring only a very small portion of T_α. The recursive procedure is as follows.

1) For each node α , use Selection to choose and order those alternatives-at- α to be explored.
2) If there are no alternatives-at- α to be explored, use Evaluation to assign a value to α .
3) If there are one or more alternatives to be explored, use Summary to assign a value to α in terms of the values found by exploring these alternatives-at- α .
4) If α is the base of the tree, than choose the alternative-at- α with the "best" value as an "acceptable" alternative-at- α . Here "best" value means that value which is the largest, or has the highest "measure of goodness."

Thus the procedure selects an "acceptable" alternative-at-a_1, say a_2. We then iteratively apply the procedure: to a_2, to find an "acceptable" alternative-at-a_2, say a_3; to a_3, to find an "acceptable" a_4; etc. At some point in this procedure the determination of some acceptable alternative will involve exploring the sub-tree to terminal nodes. Thus the last few nodes of a solution-path will be found without further search, if one exists in the sub-tree. If the final sub-tree searched in this procedure does not have a solution path, then the search has failed. This meta-heuristic procedure of using Evaluation, Solution, and Summary we will call Look-ahead.

Since Look-ahead is not an algorithmic procedure, it may fail to find a solution-path. The likelihood of its success depends on the particular special-heuristic used to perform Evaluation, Selection, and Summary. The better these special-heuristics are, the more likely it is that Look-ahead will find a solution-path.

I would like, at this point, to interject some remarks concerning the general usefulness of Look-ahead. I have not seen in the literature any clear cut explanation why it is useful at all. References to Look-ahead, as a method to be used in searching a game-tree have been made only on the intuitive basis that it seems to work in improving the performance of a heuristic program. The various theoretical arguments I have heard in private conversations with investigators in the field are, in general, specious and/or vague.

It has been argued, for example, that if the evaluation-function is very good, Look-ahead is of no practical use. In this case, the argument goes, it is sufficient to apply the evaluator to each alternative-at-α , and choose as "acceptable" the alternative with the "best" value. It is obvious that if the evaluator were perfect, this would indeed be the case. However, if we could design a perfect evaluator, the tree-searching problem would have a trivial algorithmic procedure for finding a solution, namely doing what is recommended by the argument for the case of a very good evaluator. But this argument is, in fact, false for non-trivial tree-searching problems.

Let us consider what the practical relationship is between Evaluation and Look-ahead. The evaluator provides an estimate for a node α of how likely it is that the tree-searching procedure being used will find a solution path for T_α . This is what we mean by a value in V being a "measure of goodness" for α . If the value is small it does not mean that no solution path exists for α . It merely means that it is not very likely that the tree-searching

procedure being used will find a solution path.

Thus we see that if we apply the evaluator to a node α , and obtain a value $v \epsilon V$, v is an estimate of the "goodness" of α . The actual "goodness" of α is in general different from v by an amount which depends on the goodness of the evaluator. If the evaluator is perfect, this difference will be zero. The better the evaluator is, the smaller this difference will be on the average taken over the ensemble of all elements of S. Thus v represents the mean of some distribution of values for α , and the standard deviation of this distribution varies in an inverse manner with the goodness of the evaluator. We state here, without proof, the proposition that the recursive application of Summary has the effect of assigning to the node α , a value v^* which is a better estimate of "goodness" than v, in the sense that the standard deviation of the distribution of values is reduced. A demonstration of the truth of this proposition would depend on the particular manner in which Summary is accomplished. It is in fact the case that one of the purposes of Summary is to achieve this reduction of standard deviation in estimating the "goodness" of the node.

Thus we see that the purpose of Look-ahead is to effectively use Evaluation, Selection, and Summary in a manner which provides the assigning of a value to a node with greater precision than that obtainable with Evaluation alone. In general, Selection is used so that the effort spent in Look-ahead can be in depth (to deeper plys) rather than in breadth (more alternatives explored at each node). However, in some cases, it is only with judicious use of Selection that Look-ahead can at all effectively achieve a reduction

in standard deviation. In the interest of brevity we will not include examples of such cases here.

5. Redundancy

This meta-heuristic is concerned with the means whereby the desired relationship between meta-heuristics and special-heuristics can be achieved. We recall that the nodes of the tree T correspond to the elements of the problem space S. We assert that the representation of the elements of S should be highly redundant. The intention of this assertion can be most readily understood in terms of an example. Let us consider the class of tree-searching problems associated with chess. We could, by clever coding, represent a chess position by only 168 bits. This would constitute an almost minimally redundant representation. From these 168 bits, we could, by algorithmic means, derive any desired information about the position. We could, for example, decode these 168 bits to answer simple questions such as (1) "What is on a specified square?" and (2) "Where is white's king?" We could also answer more complicated questions such as (3) "Are there any potential knight forks of king and queen?" or (4) "What units occupy the same file as white's king?" All of these questions could be more easily answered if the representation were more redundant. For example consider the more straightforward representation of 64 x 4 = 256 bits corresponding to the 64 squares times 4 bits per square to identify the occupant of the square. With this representation, finding the answer to question (1) is trivial. We might add 32 x 6 = 192 bits to the representation, corresponding to 32 units times 6 bits per unit identifying the square the unit occupies. This addition of redun-

dancy allows us to find the answer to question (2) in a trivial manner. If we continue to add redundant information to the representation, the procedure for finding answers to questions (3) and (4), as well as many other complicated questions, can be greatly simplified.

The reader might question the practicality and feasibility of maintaining a representation of sufficiently high reduncancy. That is, he might consider that the work required to generate the highly redundant representation is as much as, or more than, the work required to generate the particular data required to answer the desired set of questions. He might also consider that the space required, to store the representation for the various nodes explored by Look-ahead, will be too large for existing core memories. However, we maintain that it is only necessary to generate the changes in the representation caused by moving from a node α to an alternative-at-α . In general, the actual amount of change which takes place in such a move is small compared with the amount of information it would be necessary to generate in order to be able to answer the desired set of questions. Moreover, it is only necessary to store the changes in the representation, as we recursively move from a node α to an alternative-at-α , and we can then use these stored changes to restore the original representation when returning from the alternative-at-α to the node α .

Let us consider how the various questions we ask about a node relate to the organization of a heuristic procedure for tree-searching. The answers to these questions about nodes are used by the various special-heuristics which perform Evaluation and Selection.

In fact, these special-heuristics are to be defined in terms of the answers to such questions. Thus we see that there is a significant advantage to be obtained from the use of a highly redundant representation of the elements of the problem space S, namely, the advantage of facilitating and speeding-up the computation required to perform Evaluation and Selection, and thereby the entire recursive procedure of Look-ahead.

There are other advantages gained besides ease and efficiency of computation. Since we anticipate the desirability of adding special heuristics to the procedure in order to improve performance, the fact that the representation is highly redundant will preclude the necessity of preparing programs to generate the information needed by the special-heuristic in answering the questions, in terms of which the special-heuristic is defined. Thus we gain facility in incorporating special-heuristics into the heuristic program. In addition, the fact that we describe a special-heuristic in the form of a function of the representation allows us to express or describe the special-heuristic in a simple canonical form. This provides us with what is essentially a compiler language for representing special-heuristics.

Thus we see that there is a three-fold purpose in using a highly redundant representation of the elements of the problem space S.

1) Providing a language in which we can represent special-heuristics in a simple manner.

2) Facilitating the organization of special-heuristics into

the heuristic program.

3) Speeding up the computation required to accomplish Evaluation and Selection.

The meta-heuristic of using a highly redundant representation to accomplish this three-fold purpose we will call Redundancy.

6. Planning

The purpose of this meta-heuristic is to improve the performance of Selection. The special-heuristics used in applying Planning to a particular class of tree-searching problems are to be designed so that sub-goals will be generated which Look-ahead will attempt to satisfy. By a sub-goal we mean the goal of finding a node, by means of Look-ahead, about which a specific question is answered favorably.

Let us here use chess as an example of the use of Planning. We assume that there will be special-heuristics used to answer a sequence of questions about the node α . One such question might be, "Do I have an open center file not occupied by one of my rooks?" If the answer to this question is yes, the sub-goal is generated, "Find a node such that I have a rook on an open center file." In addition to having this sub-goal generated, there will be plausible-move-generators which will generate a sequence of moves for bringing about this sub-goal. The manner in which these objectives will be accomplished will be discussed in more detail when the various aspects of Phase 1 of the proposed research are presented.

We will call Planning the meta-heuristic of choosing, in this manner, questions to be answered and plausible-move-generators

which try to answer these questions, where the choice of these questions and generators is made on the basis of the properties of the base of the sub-tree being explored by Look-ahead.

The generality of the heuristic theory.

We have presented a theory of heuristic methods of tree-searching in terms of meta-heuristics and special-heuristics. As part of this theory we have described a heuristic procedure expressed in terms of the six meta-heuristics Evaluation, Selection, Summary, Look-ahead, Redundancy, and Planning. We now wish to assert the generality of this procedure.

We recall that in presenting the descriptive theories of trees and problems, we noticed a clear distinction between competitive and non-competitive situations. We also recall that when we discussed algorithmic methodology applied to tree-searching, the theoretical distinction between competitive and non-competitive trees was no longer of any practical significance. We now observe that in terms of the heuristic theory of tree-searching, this distinction is non-existent. It is of course true that the special-heuristics used to perform the various meta-heuristics will be different in a program for searching non-competitive type trees from the special-heuristics used in a program for searching competitive type trees. However, this difference in special-heuristics exists in the same sense when considering the competitive trees for chess and the competitive trees for checkers, as well as when considering the non-competitive trees for theorem proving and the non-competitive trees for sentence parsing. Thus the heuristic pro-

cedure described in the heuristic theory is, theoretically, uniformly applicable to a very general class of well-defined classes of tree-searching problems.

The proposed research: Introduction.

In these final sections, we will present a discussion of each of the four phases of the proposed research. The research will involve a practical application of the heuristic theory. The heuristic procedure described in the heuristic theory is to be applied to a chess playing program.

The game of chess was selected as the immediate object of study in this research for several reasons. Firstly, the tree of positions associated with a given chess postion are typical of the large, highly structured trees which constitute the paradigm of the problem area. Secondly, superiority of chess playing performance of humans seems to be heavily dependent on superior skill in selecting branches to investigate, and in evaluating the resulting positions. These evaluations are, in effect, an estimate of the summary of information for branches below the node being evaluated. Thirdly, chess was chosen for the study because of the author's strong interest in and knowledge of the game. We also note that several of the outstanding workers in the area have agreed that chess presents a particularly attractive combination of features and problems.

Phase 1: The heuristic program

This phase of the research will involve the writing and debugging of a program to play chess. The program will be organized in the manner of the heuristic procedure as described in the heuristic

theory. We will present a description of how each of the six meta-heuristics previously discussed will be incorporated into the chess playing program.

1. Evaluation

The special heuristics to be used as evaluators will be designed more or less according to the ideas presented in Point Count Chess [2]. The values assigned to the fighting units will be approximately: pawn = 3, knight = 9, bishop = 9, rook = 15, and queen = 27. Special heuristics will be written which ask questions about the position being evaluated. The purpose of these questions is to ascertain the presence of various favorable or unfavorable features. Each feature present will be worth one point, i.e. one-third of a pawn. The score for each player, white and black, will be the sum of the scores for his fighting units plus the number of favorable features present in the position minus the number of unfavorable features. The value for the position is the difference between the two scores, one for each color.
Point Count Chess asserts that a score of +4 is a theoretical win.

In this reference about 30 features are described. Examples of favorable features are: pawn on fourth rank vs. pawn on third rank, outpost for a knight, control of open file. Examples of unfavorable features are: isolated pawn, bad bishop, poor king position. In some cases it will be impossible to answer accurately a particular question about the position being evaluation. In this case Selection will generate sequences of moves for the particular purpose of answering the question. This provides us with what Point Count Chess calls "dynamic evaluation". The manner in which

Selection performs this function will be described in the next sub-section.

2. Selection

The plausible-move-generator will be organized as follows. There will be four subroutines called by the plausible-move-generator. Each will generate a specific kind of move for each color to be tried at the given position of the sub-tree being explored. The four kinds of moves will be:

(a) Tactical threats and defenses;

(b) Strategical plans: offensive and defensive;

(c) Moves to assist in dynamically evaluating a feature of a given position;

(d) Moves to improve a previously analysed variation using the results of this analysis as input.

(a) The tactical-threat-generator will generate moves to effect or defend against a particular set of tactical threats. Typical members of this set are: knight fork, pin, threaten piece with pawn, threaten major piece with minor piece.

(b) The strategical-plan-generator will operate as follows. At the base of the sub-tree (the input position) this generator will evaluate the position with respect to a set of positional features. These features, if present, are either favorable of unfavorable. For example, control of open file, pawn on 4th vs. pawn on 3rd, bishop vs. knight on an open board, etc. are favorable. Doubled pawns, king kept in center against will, bad bishop, etc. are unfavorable. For each feature two sequences of moves will be

generated, one for each color, which will attempt to effect a favorable change of the position with respect to the feature from the viewpoint of each of the two colors. At deeper plys of the sub-tree the generator will select the next move of the sequence, or perhaps initiate a permutation of the sequence. By applying Look-ahead in this way the best plan for each color will be selected. The selected plans will be attempts to improve the position with respect to some specific characteristic from the point of view of each of the two colors. The best plan for each color will then be used in a combined look-ahead procedure in order to evaluate which plan is more important. In this way the final decision will be made as to which single plan the program should use in deciding on its move. This final plan will either be an attempt to effect the computers best plan, or to inhibit the opponents best plan. It should be kept in mind that during the application of Look-ahead the other three plausible move generators will be active. Of particular importance will be the tactical-threat-generator, so that, in the attempt to effect a strategical goal, tactical possibilities will be considered. It is the use of the strategical-plan-generator which accomplishes the meta-heuristic of Planning.

(c) The dynamic-evaluation-generator will be used to assist in deciding whether or not a specific feature is present in the position being evaluated. This may require extending the tree in a very limited fashion from this position. For example, in order to decide whether white should be credited with an outpost for his knight, Selection is used to extend the tree. While exploring this

extension of the tree, a test is made to determine if the knight is able to occupy the outpost, and if the opponent is unable to prevent or terminate this occupation. This means that if the opponent tries to interfere with the knights occupation of the outpost, he will incur another weakness which is at least potentially countable as a favorable feature for white.

(d) The improvement-generator will have a very limited by important function. When an exploration of a sub-tree below a given node indicates that a specific move has been effective in refuting an otherwise strong variation at many nodes of this exploration, then at the given node, a move is generated to prevent the specific refuting move and the variation is re-evaluated.

3. Summary

The heuristic used to summarize the values below a given position into a value for the position will be as follows. There will be a standard min-max evaluation. There will also be a weighted average taken. The weights will depend on the effort involved in making the evaluation. The exact nature of this weighted average will be determined by experimentation in phase 2. What this procedure will attempt to effect is that the program should vary its play according to the evaluation. That is if it evaluates the position as a won game it should use the min-max value and play conservatively. If it has a lost game, it should use the weighted average in a manner such that it plays for a position where the opponent is most likely to make a mistake.

4. Look-ahead

This meta-heuristic will be programmed in a straight forward manner as a recursive program. As the exploration of the tree recurses from a position (node) to an alternative at the next ply, the changes in the representation of the position are stored. That is when a particular parameter of the representation is to be changed, the old value of the parameter is stored on a push-down list along with the location of the parameter in core. When returning from this ply, the old values are restored into their former locations, and the representation of the position is re-established.

5. Redundancy

It is expected that the representation of the position will involve approximately 2000 words of 36 bits each. The representation will primarily include bit patterns, (two words for each 64 bit pattern), each bit pattern representing those squares which have a common characteristic. The set of such characteristics will contain such items as:

1. Those squares guarded by black pawns;
2. Those squares a knight move away from two white pieces;
3. Those squares containing pinned units;

etc.

6. Planning

This meta-heuristic is included in the program in the form of a subroutine to the plausible-move-generator described under Selection, namely the strategical-plan-generator. It is understood that the evaluator will also be written in such a way that the answer

to a particular question can be ascertained and perpetuated by Summary.

Phase 2: Experimentation

This phase of the research will entail having the computer play about twenty games of chess under various conditions. It is not expected that twenty games will be sufficient for a complete and definitive evaluation of the various techniques being investigated. However, it is not practical at this time to anticipate having the computer play many more than twenty due to the limitations of available 7090 time. In spite of this limitation of about twenty games, it should be possible to indicate, in a general way, the advantages of the techniques (i.e. the meta-heuristics) and the kinds of experiments useful in making a definitive study were adequate computer time available.

In particular it is expected to run several experiments of each of the following kinds.

1. Computer vs. Human

These experiments should be the most interesting from the point of view of evaluating the general overall performance of the program under various conditions. In particular, it will be desirable to have humans play the computer using a configuration of the program intended to achieve the best possible performance. In addition, it will be useful to try various configurations which would weaken the overall performance but would allow the computer to decide on its moves faster. In this way more games could be played, albeit not as good games, and the program's performance could be evaluated

under "rapid transit" conditions, where human performance also deteriorates somewhat.

2. Computer vs. Computer

A series of various configurations will be played, one against another, in order to evaluate the effects of various programmed chess special-heuristics. One variable of these alternative configurations can be used in the manner of a control, or scale, in comparing the effects of the different heuristics. This variable is time, or more accurately, total effort allowed in analysis. Thus we might compare the performance of two configurations, one having included an additional special-heuristic, and the other being allowed more time to consider its moves. It is not intended that these experiments should indicate the practicality of particular special-heuristics, but rather that a demonstration be made of the possibility of making a reasonable comparison of the relative worth of the special-heuristics used in a heuristic program.

3. Computer vs. Book

For this series of experiments the computer will play over book games of the masters. From these experiments it is intended that particular weaknesses of the program can be pinpointed by discovering the reasons the computer missed or avoided lines recommended by the masters. The organization of the program (i.e. Redundancy) should allow for convenient and simple additions of special-heuristics to eliminate these pinpointed weaknesses. This is certainly expected to result in improving the computer's overall performance. More important, however, will be the demonstration of the advantages of Redundancy in attacking the problem of recog-

nition and repair of weaknesses in heuristic programs by means of the addition of new heuristics.

Phase 3: Evaluation of Experimental Results

As indicated above, the playing of about 20 games of chess will not allow for making a definitive evaluation of the meta-heuristics being tested. However, this phase will serve the purpose of evaluating, at least in a general way, whether or not the meta-heuristics are useful in achieving the desired goals. In addition, it will be ascertained, whether or not it is plausible to assume that an extended series of experiments would permit a definitive evaluation.

Phase 4: The Report

The report on this research will include:

1. A detailed description of the program
2. The evaluation of the experimental results
3. A general review of work in the field
4. A general discussion of the present research as compared with earlier work in the field
5. A detailed description of the experiments including the motivation for the particular experiments.

References

[1] Edwards, D. J. and T. P. Hart, "The Tree Prune (TP) Algorithm," Memo 30--Artificial Intelligence Project--RLE and Computation Center, MIT, Cambridge, Mass. December 4, 1961.

[2] Horowitz, I. A. and G. Mott-Smith, Point Count Chess, Simon and Shuster, New York, 1960.

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM-47

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)

☐ Other:_____

Document Information

Number of pages: 32 (36-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:

☒ Single-sided or

☐ Double-sided

Intended to be printed as :

☒ Single-sided or

☐ Double-sided

Print type:

☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: MIMEOGRAPH

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): _____

Photographs/Tonal Material (by page number): _____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP:	(1-32)
TITLE PAGE,	1-3
SCANCONTROL TRGT'S	(33-36)

Scanning Agent Signoff:

Date Received: 11/30/95 Date Scanned: 12/1/95 Date Returned: 12/14/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**